Advanced R: A Gentle Intro to Statistical Learning

Jared Berry

January 21, 2020

A word on data science and economics

The training you are all getting in econometrics and regression analysis makes you a tremendous asset to employers on the job market

- For those of you looking to explore work (or even unique research questions) outside the purely academic economics niche, knowledge of open-source tools (e.g. R, Python) and a handle on the basics of data science/statistical learning can open a lot of doors
- Most "data scientist" (or even analyst) roles will expect, and test, a level of expertise using R or Python along with knowledge of how to build and evaluate a predictive model

My goals

- Provide a rough foundation in statistical learning/predictive analysis using R
- Frame the rigorous training you all have in econometrics and regression analysis in a predictive, rather than inferential, context
- Walk-through examples of setting up models to perform predictions
- Learn methods for evaluating those predictions
- Explore, at a high-level, some statistical learning models outside the scope of traditional econometric training (OLS)
- Offer some degree of experience and comfort doing predictive analysis
    - What you will not be after this class: an expert on machine/statistical learning
    - What you will be after this class: comfortable enough to set up a rudimentary predictive model, explain why you picked it, and evaluate its performance

Today

- Statistical Learning
  - Estimating f
  - Prediction v. Inference
  - Prediction Accuracy v. Interpretability
  - Regression v. Classification
  - Assessing Model Accuracy and the Bias-Variance Tradeoff
- Linear Regression (OLS)
- Logistic Regression
- Cross-validation

Beyond

- Cross-validation (cont. . .)
- Other models
  - Lasso/Ridge
  - Decision Trees
  - Random Forest (More trees)
- Deliverable

Overall

We've branded this as "advanced R", largely due to the subject matter and importance of having a foundation in R, but this will skew more concept heavy than programming heavy

- Ideal: High-level conceptual understanding, coupled with practical application
- It does *not* take much code to run these (or really any) models - typically just a couple of lines
- The rule of thumb is that 80% of a project is spent preparing (sourcing, ingesting, cleaning, etc.) data, and just 20% is spent modeling/visualizing

What is "statistical learning"

- At the simplest level, we want to use some function $f(X)$ to predict an output variable Y using some set of input variables X.
    - i.e. is there some function that relates sales to advertising through different media
    - Terminology for these things differs
    - Inputs are sometimes called predictors, independent variables, features, variables, covariates etc.
    - Output is generally called dependent variable, response, target, etc.
- We typically represent this as:

$$y = f(X) + \varepsilon$$

    - Where ε represents random error
- In statistical learning, econometrics, social science in general, etc. we are interested in estimating the function f
    - The regression equation you are all so intimately familiar with is the simplest way to do this!

Why would we want to do this

Prediction

- Predict $\hat{Y}$ given some fitted function $\hat{f}(X)$ and predictors X (i.e. $\hat{Y} = \hat{f}(X)$ - hats denote fitted/predicted values)
  - We want to maximize the *accuracy* of $\hat{Y}$ as a prediction for Y
  - This depends on both *reducible* and *irreducible* error

$$E(Y - \hat{Y})^2 = E[f(X) + \varepsilon - \hat{f}(X)]^2$$
$$E(Y - \hat{Y})^2 = [f(X) - \hat{f}(X)]^2 + Var(\varepsilon)$$

- Where $E(Y - \hat{Y})^2$ represents the *expected value* of the squared difference between the predicted and actual value of Y, and $Var(\varepsilon)$ represents the *variance* associated with the error term e.
  - The first portion of the right side of this equation represents *reducible* error, while the second portion represents *irreducible* error
  - We're interested in finding/estimating f with to minimize the reducible error
  - Less concerned about standard errors in making predictions
  - Not quite "forecasting" which is more a pure time-series exercise

Why would we want to do this

Inference

- Rather than focusing on the accuracy of *predictions* of $\hat{Y}$, we might want to understand the *relationship* between X and Y, or understand how Y changes as a function of $X_1, ..., X_p$
  - This is the more traditional use case for the econometric work you are all learning, and necessitates correct standard errors (i.e. robust)
  - Questions like "Which predictors are associated with the response?" and "What is the relationship between the response and each predictor?"
  - Best answered using highly-interpretable (i.e. linear) models, which we will diverge from later on

How do we estimate f?

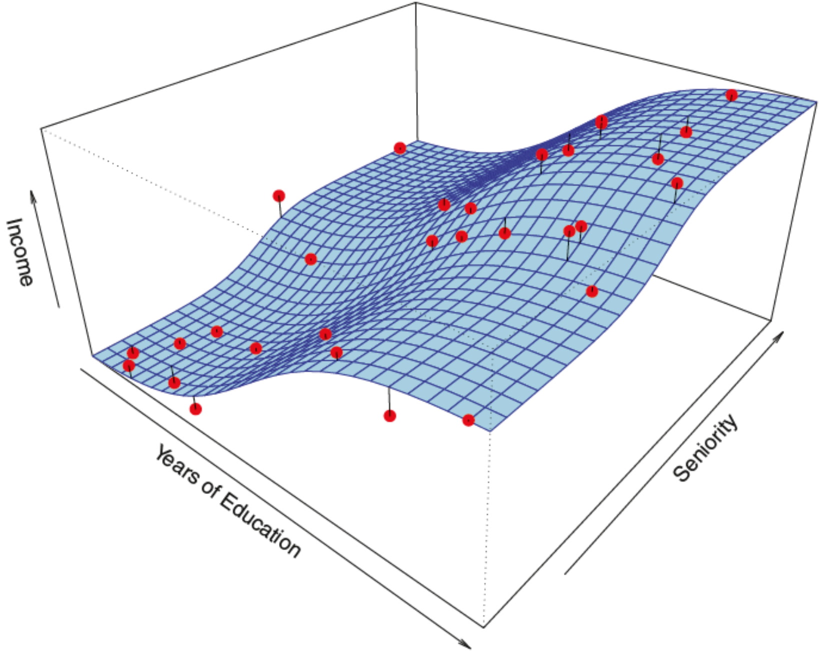
- Our goal is to apply some sort of estimation/statistical learning method to *training data* to estimate f, which is unknown.
  - We rarely get to know what function actually dictates the relationship we're estimating
  - Instead, use some estimated function $\hat{f}$ where $Y \approx \hat{f}(X)$
- *Parametric methods* involve reducing the estimation of f to estimating a set of parameters, i.e. coefficients in a linear model

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$$

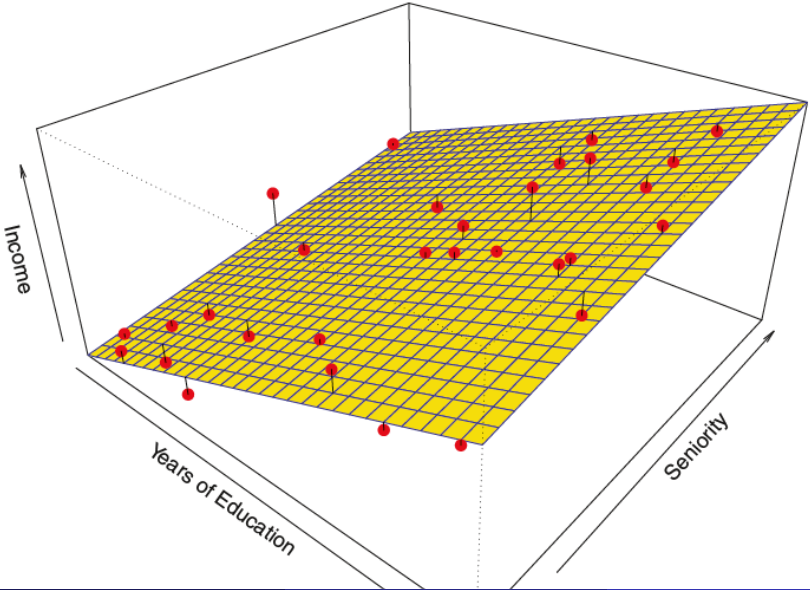
- Here, we would use our training data to *fit* or *train* the model, typically using ordinary least squares
- This approach generally oversimplifies reality (reality is probably *not* linear), but generally provides a good, *interpretable* approximation of $\hat{f}$

How do we estimate f?

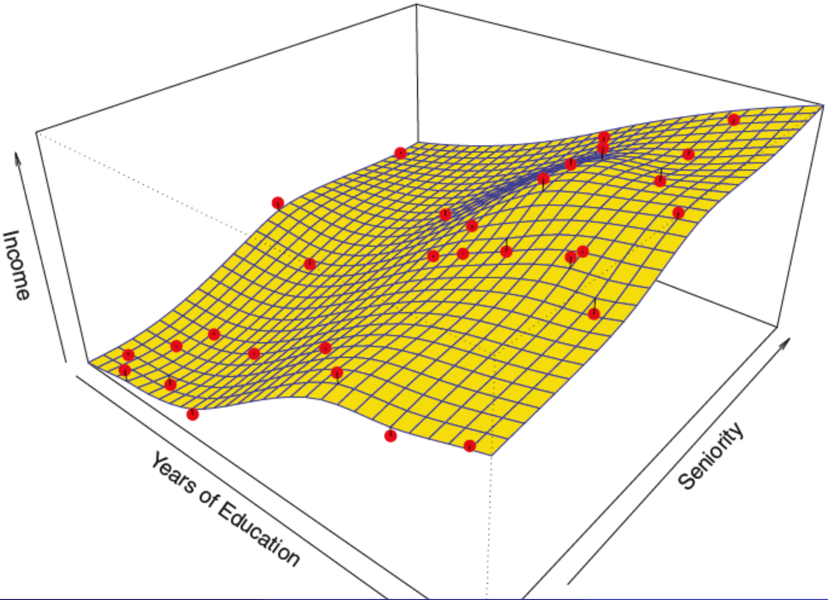
- *Non-parametric methods* make no assumption about the functional form of f and instead estimates an arbitrary function f that gets as close to the actual data points as possible
  - Far more flexible but requires a *very* large number of observations
  - Far *less* interpretable


Income
Years of Education
Seniority

Using a linear model (parametric)



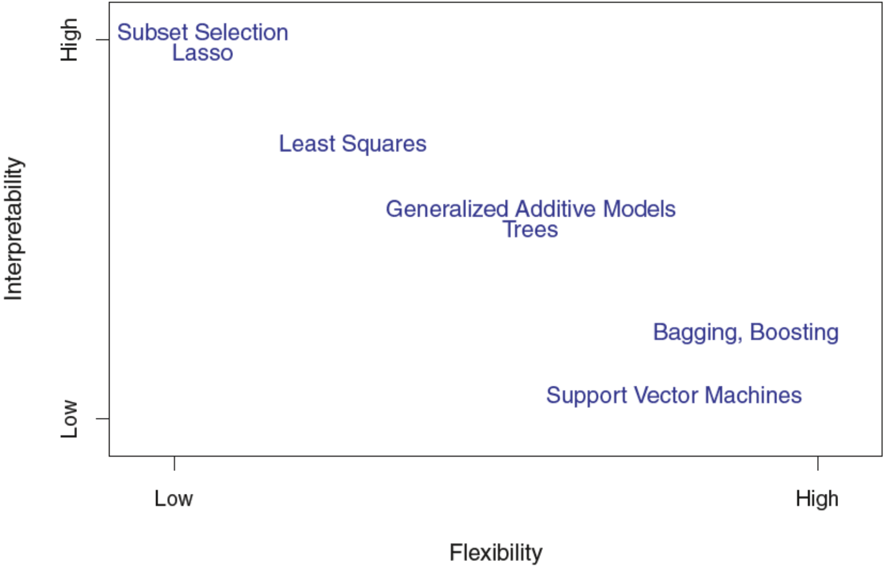
Using something non-parametric



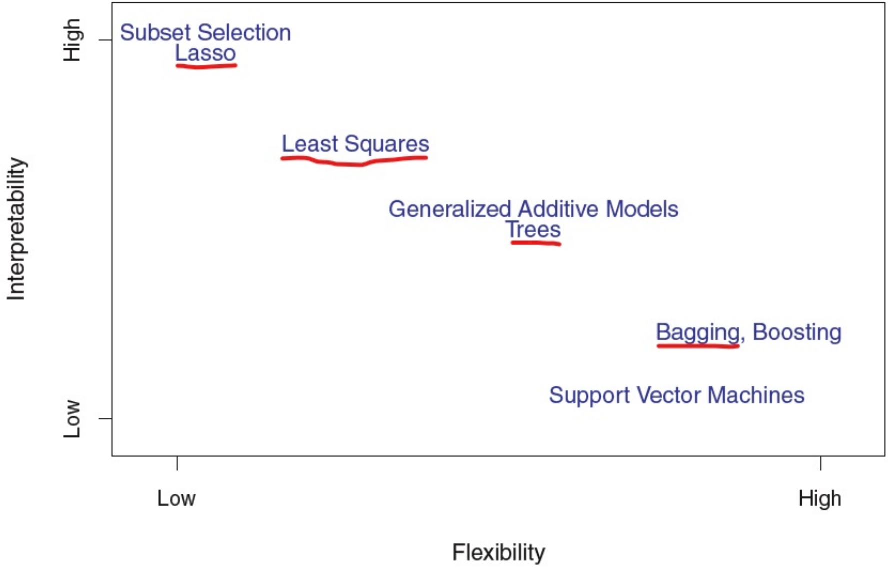
Trade-offs

- Prediction accuracy (as a function of flexibility) vs. interpretability.
  - Linear models are easy to interpret; thin-plate splines (in the third image - we will *not* talk about these) are not.
- Good fit vs. overfitting vs. underfitting.
  - How do we know when the fit is just right?
- Parsimony vs. black-box.
  - Simpler models involving fewer variables (sparse models) vs. black-box predictors involving everything.
  - Closely related to the interpretability trade-off

Flexibility vs. Interpretability



Flexibility vs. Interpretability



Regression vs. Classification

- Variables can be *quantitative* (numerical values) or *qualitative* (one of a number of categories, in other words, *categorical*)
    - We typically think of quantitative dependent variables as the subject of *regression* problems (ordinary least squares regression)
    - We typically think of qualitative dependent variables as the subject of *classification* problems (logit/probit models, or in this context, *logistic regression*)
    - We will look at some models that can do both

Measuring quality of fit (regression)

- In the regression setting, want to minimize *mean squared error* (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

- i.e. what is the average squared difference between our function f's prediction for the ith observation, and the observed value y_i
  - We'll also look at *RMSE* (*root* mean squared error) which is the square root of this value (better maps to units on the y-axis)
  - *MAE* (mean absolute error) is a well-accepted metric as well (minimizes outliers)
  - These will be reiterated in time-series econometrics/forecasting

Measuring quality of fit (regression)

- We can compute these metrics (MSE, RMSE, MAE) on our training data, but what we're *really* interested in is how well we can fit *unseen test data*
  - If we trained a model/algorithm to predict stock prices using the past 6 months of returns, we'd want to know how well it could predict *tomorrow's* returns
  - How well our models perform on data we have through today is far less interesting than how well we can predict *future*/*new* outcomes
- We are also likely to *bias* our model by evaluating it's performance on the training data by *overfitting*
  - Chasing individual data points in the training data is bad if our data change

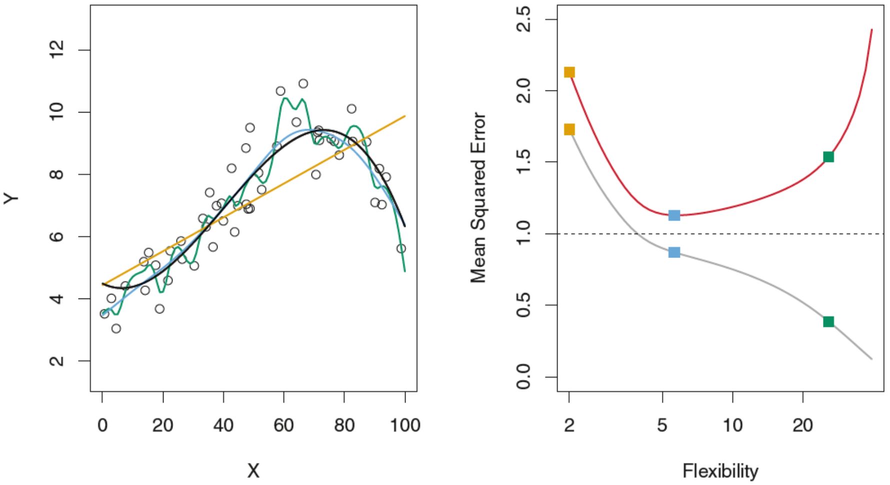
Measuring quality of fit (regression)

- We're really interested in minimizing *test* MSE. If (x_0, y_0) is an unseen test observation not used to train the model, we want a model that minimizes:

$$Ave(\hat{f}(x_0) - y_0)^2$$

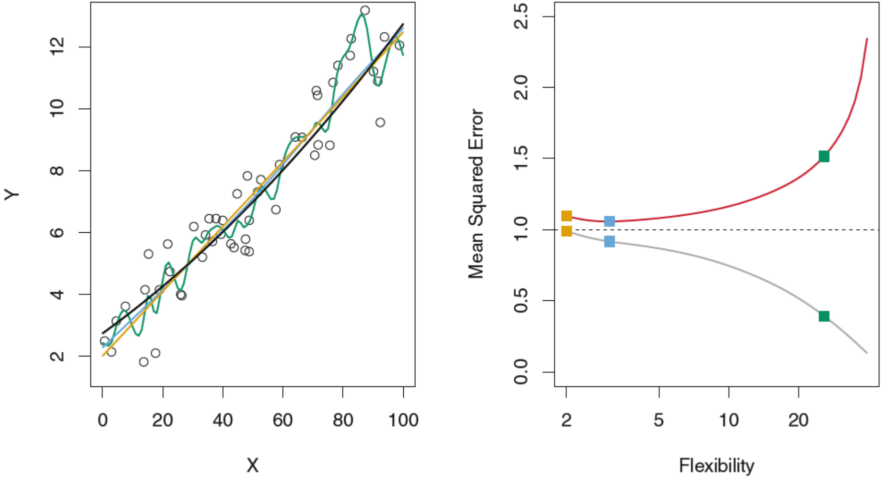
for all *test* observations

- In practice, getting test data is difficult (can't get tomorrow data today!)
  - We don't necessarily just want to pick the model with the lowest training MSE
  - What if our model fit on training data doesn't map well to test data?
  - We'll discuss a method for getting appropriate test data called *cross validation*
- This tension between training/test performance is captured in the *bias-variance tradeoff*

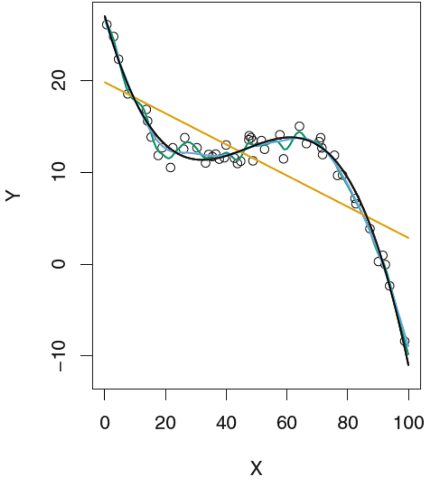


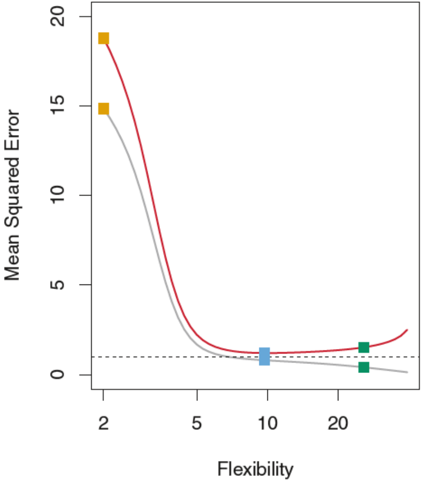
Y
X
2
4
6
8
10
12
0
20
40
60
80
100
Mean Squared Error
Flexibility
0.0
0.5
1.0
1.5
2.0
2.5
2
5
10
20

The Bias-Variance Tradeoff



The Bias-Variance Tradeoff





The Bias-Variance Tradeoff

Test MSE can be decomposed into three parts, the *variance* of $\hat{f}(x_0)$, the *bias* of $\hat{f}(x_0)$, and the variance of the error term ε:

$$E(Y - \hat{Y})^2 = Var(\hat{f}(x_0)) + [Bias\hat{f}(x_0)]^2 + Var(\varepsilon)$$

In simple terms...

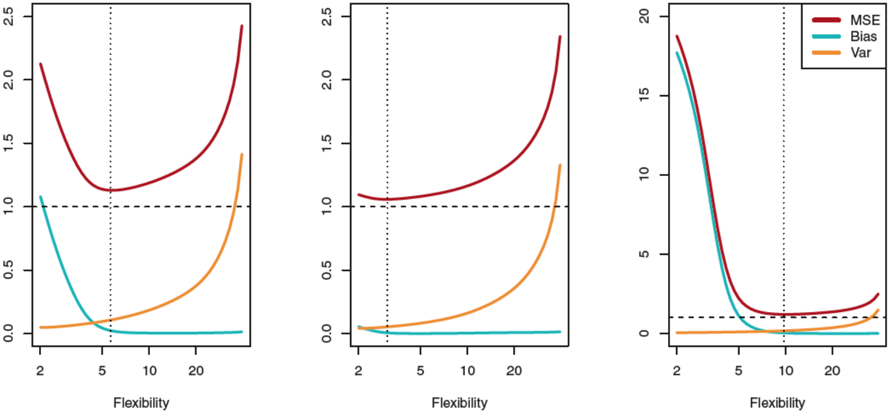
- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated the function using different training data
  - A method with high variance means that very small changes in training data significantly impact the method used to estimate it
- *Bias* refers to the error that is introduced by approximating a (potentially extremely complicated) *real-life* problem using a simple model
  - It's highly unlikely any *real world* problem can be truly approximated using a simple linear model

The Bias-Variance Tradeoff

Typically as the *flexibility* of $\hat{f}$ increases, its variance *increases*, and its bias *decreases*.

- The more complicated our model is, the better we'll be able to get at the complexity of the problem we're seeking to approximate
- The degree to which we chase individual data points to accomplish this, means that diverging from our test data can lead to disastrous performance
- Choosing the flexibility based on average test error amounts to a bias-variance *trade-off*

The Bias-Variance Tradeoff



Measuring quality of fit (classification)

- All of the above applies, but how do we evaluate the performance of our models in a classification (qualitative) context?
- There are a number of ways to do this, simplest being *error rate*, or the proportion of mistakes made when we apply $\hat{f}$ to our data:

$$ErrorRate = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

- Where I is an indicator variable that is 1 if $y_i \neq \hat{y}_i$, and 0 otherwise
- Given test observations of the form (x_0, y_0), minimize the test error rate associated with:

$$Ave(I(y_0 \neq \hat{y}_0))$$

- We'll also assess prediction *accuracy* or the proportion of labels correctly categorized

Simple Linear Regression (OLS)

- Assume a model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- By estimating $\hat{\beta}_0$ and $\hat{\beta}_1$ we can generate predictions for our target $\hat{y}$
- Using least squares, we generate estimates of $\hat{\beta}$ by minimizing residual sum of squares (RSS), where $e_i = y_i - \hat{y}_i$ represents the ith residual and

$$RSS = e_1^2 + e_2^2 + \cdots + e_n^2$$

Simple Linear Regression (OLS)

- Least squares minimizing values are found to be

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- We can assess the accuracy of our coefficient estimates by generating standard errors and t-statistics in the usual way (no need to derive these here)
- We can assess the accuracy of our model as a whole by examining *Residual Standard Error* or R^2
- We can introduce additional features into the model (multiple linear regression), provided we keep straight how to interpret those coefficients
    - *ceterus paribus*

Simple Linear Regression (OLS)

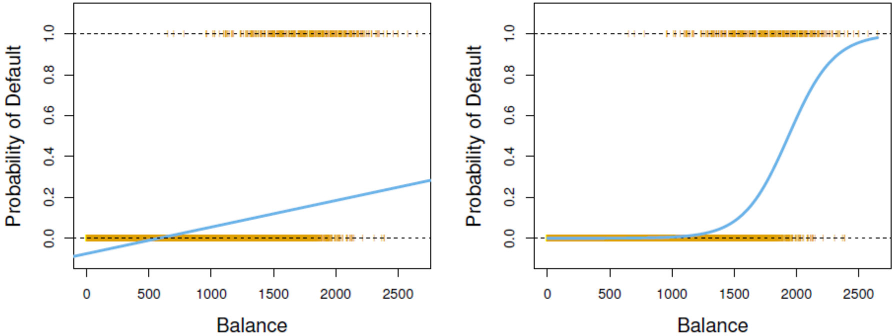
Despite the fact that our features *enter* the model linearly, we are not restricted to purely linear features

- Qualitative predictors (dummy variables)
  - *All* of the assumptions from QMI still apply!
  - Remember your no perfect collinearity assumption! One level must be dropped
  - `model.matrix` is a useful tool for encoding character/factor variables as dummies!
- Interaction terms (remember the hierarchy principle)
- Polynomial features

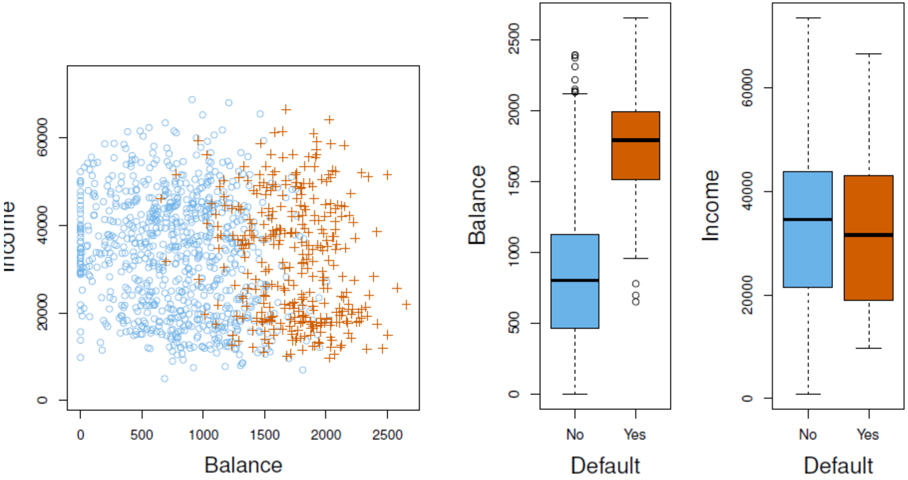
Review your notes from QMI and QMII for all of the above

Classification

- When we have *qualitative* response variables, we are no longer interested in predicting numerical values associated with Y, but instead some function f that estimates *probabilities* X belongs to one of a set of categories
  - i.e. What is the probability an individual will default on their credit card balance? What is the probablity a certain transaction is fraudulent? What is the probability a stock price will move up tomorrow?
  - How to estimate? We could use linear regression (linear probability model), but it's not well-suited for binomial targets



Classification



- What we're looking for is a set of predictors that can *separate* out our classes
  - This notion of *separability* is part and parcel of performing classification tasks

Logistic (Logit) Regression

- Models the *probability* Y belongs to a category, so in the Default $\sim$ Balance example above, $Pr(default = Yes|balance)$ (or just $p(balance)$).
  - From predicted probabilities we can assign a *class label* (i.e. above 50%)
- We now frame our regression model as

$$p(X) = \beta_0 + \beta_1 X$$

- Linear probability models (LPMs) fitting a straight line to a binary response variable will result in predicted probabilities less than 0 or greater than 1!
- Enter the *logistic* or *sigmoid* function, which is an S-shaped curve that compresses values between 0 and 1

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

Logistic (Logit) Regression

- So this...

$$p(X) = \beta_0 + \beta_1 X$$

- Becomes...

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

- And with a bit of manipulation...

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X}$$

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X$$

Logistic Regression

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

- We estimate this thing using *maximum likelihood*
  - Intuition: seek coefficient estimates such that the predicted probability of default is as close to what we actually observe as possible
  - Consult your QMII notes
- Recall that we *cannot* interpret the coefficients from this model straightforwardly
  - The thing on the left is the *log-odds* or *logit* - increasing X by one unit changes the *log-odds* by β_1
  - Even then, since this isn't a straight line, probability estimates depend *where we are on the line* hence all the margins nonsense in QMII
  - Statistical significances/signs still apply for interpretation

Logistic Regression

```r
fit <- glm(default ~ balance, data=ISLR::Default, family = binomial)
lmtest::coeftest(fit)
```

```
##
## z test of coefficients:
##
##                Estimate  Std. Error z value  Pr(>|z|)
## (Intercept) -1.0651e+01  3.6116e-01 -29.492 < 2.2e-16 ***
## balance      5.4989e-03  2.2037e-04  24.953 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Making predictions

- In order to generate probabilities of default, we use the *inverse logit* and plug in values:

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}} = \frac{e^{-10.6513+0.0055\times 1000}}{1+e^{-10.6513+0.0055\times 1000}} = 0.006$$

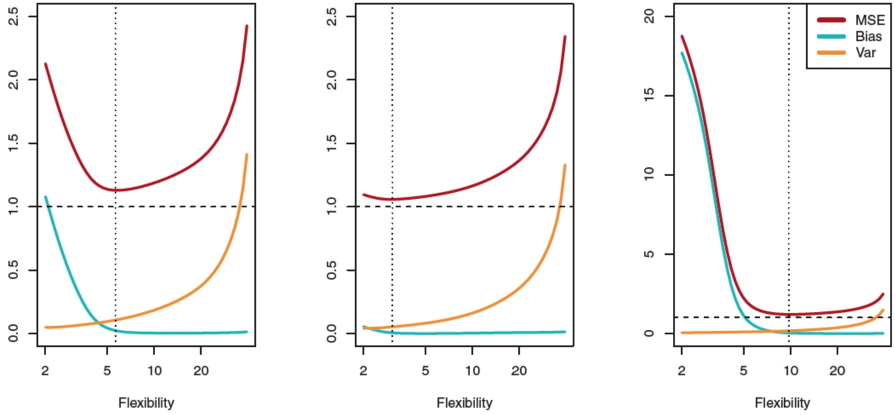
- Using these predicted probabilities, we can generate *predicted classes*, and evaluate the performance of the model using metrics like *accuracy*
  - Build a *classification report* to see how the model is actually assigning classes
  - Other common metrics include ROC/AUC curves and Precision-Recall curves (outside the scope of this session)
- All of the above applies with multiple logistic regression

Training and Test error

Recall that, in the predictive setting, there is a critical distinction between *test error* and *training error*

- We're really only interested in how well our models are performing out-of-sample (on unseen test data)
- Training error rate is likely to be different from test error rate, and more specifically is bound to *underestimate* it

Training vs. Test performance



Training and Test Error

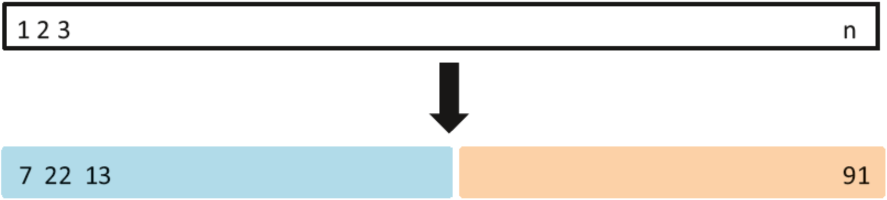
What is a practitioner to do?

- Ideally, we'd like a designated test set we could take our trained model to and evaluate against
  - In practice, this is usually infeasible
  - Some evaluation metrics (e.g. Adjusted R^2, AIC, BIC, etc.) make an adjustment for this; we will not discuss them here (covered in time-series).
- What we *can* do, is *hold-out* a subset of our training data to use as test data

Validation Set Approach

Create a *hold-out* or *validation* set from training data to compute an estimate of the test MSE (how would the model perform on unseen testing data)

- Randomly subset the data and partition off the validation set
- Train a model on the portion of the data *not held out for validation*
- Generate predictions using the observations in the validation set
- Compute some estimate of test error (MSE/RMSE for regression, misclassification rate/accuracy for classification)



Drawbacks

- Estimates of test error will be highly dependent on the observations held-out
- Using only a subset of the observations to train the model means that test error computed on hold-out set might actually be an *overestimation*
  - Less data = less accurate estimates
- Cross-validation helps to address these limitations

K-Fold Cross-Validation

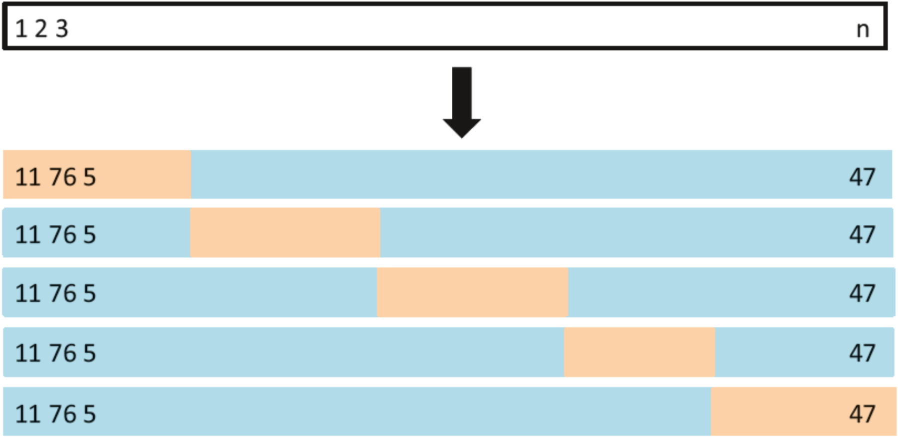
Industry standard approach for approximating the performance of a model out-of-sample

- Widely used
- Helpful for determining the best model for making predictions out of sample
- Provides a good approximation for how that "best model" will truly perform on unseen test data

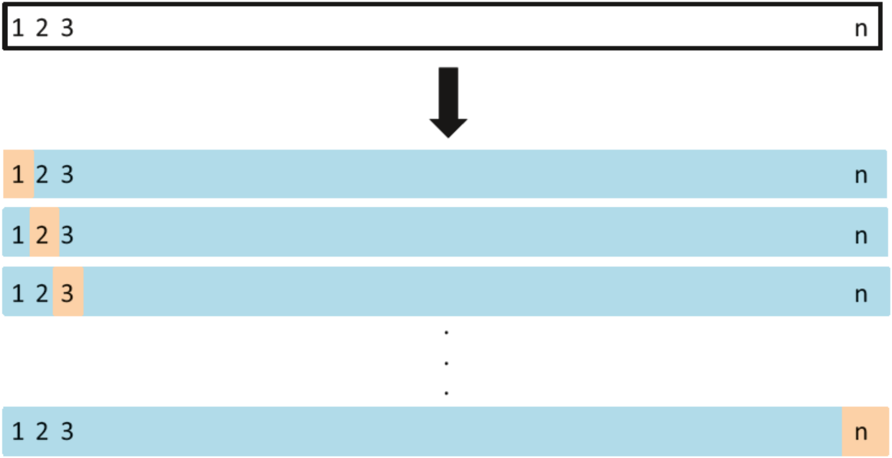
Think multiple validation sets

- Divide up the data into K equal parts
- Fit a model on $K - 1$ of these parts, generate predictions on the k-th part, and compute test error on that portion
- Repeat for each part $1, ..., K$
- When $K = n$, it's a special case called *leave-one-out cross-validation* (LOOCV)
- Best practice dictates using 5-10 folds
    - Too *many* folds actually reintroduces variance - any idea why?

K-Fold Cross-Validation



LOOCV



Cross-Validated Test Error

Let the K parts be $C_1; C_2; ...; C_K$, where C_k denotes the indices of the observations in part k. There are n_k observations in part k: if N is a multiple of K, then $n_k = n/K$. - Our cross-validated test error is the following:

$$CV(K) = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k$$

- where

$$MSE_k = \sum_{i=1}^{C_k} \frac{(y_i - \hat{y}_i)^2}{n_k}$$

- The same principles apply for classification, but with a different error metric (misclassification rate, for instance)
- *Be very careful doing this with time-series data* - any idea why?

Shrinkage Methods and Regularization

- There are ways we can *improve* the performance of our linear/least squares models on both the predictive and inferential dimensions
  - Models with tons of predictors that "chase" individual training data points might have poor fit out-of-sample (variance!)
  - Models with tons of predictors are going to be far more difficult to interpret, and it's possible we might be including variables that are irrelevant
- Shrinkage methods (or regularized models) can help us perform *variable selection* to arrive at an optimal subset of our overall set of predictors
  - These models will *shrink* our coefficients toward zero, which can reduce variance
  - Recall that *variance* refers to the amount by which $\hat{f}$ would change if we estimated the function using different training data

Ridge Regression

- In least squares, we pick coefficients by minimizing RSS for p predictors:

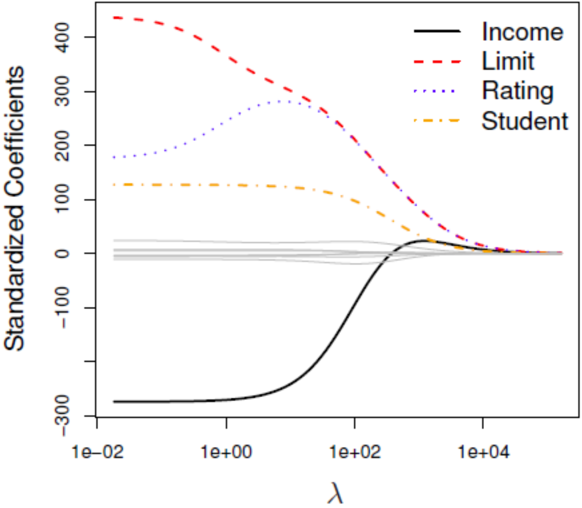
$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} B_j x_{ij} \right)^2$$

- In *ridge regression*, we minimize a somewhat different quantity, by adding a *shrinkage penalty* and *tuning parameter* λ

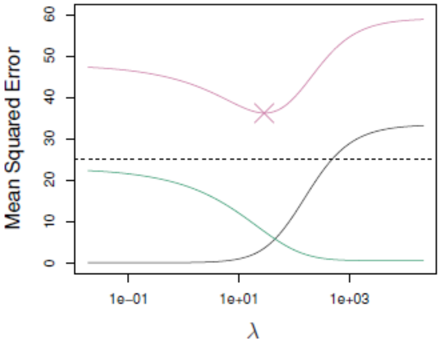
$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} B_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

- The second term (the shrinkage penalty) is *small* when $\beta_1, ..., \beta_p$ are *small*, and λ determines how much the size of the cofficients should be penalized
- We'll use *cross-validation* to pick an optimal value of *lambda*

Ridge Regression using Credit data



Ridge Regression using Credit data



- Will perform best when least squares estimates have high variance (getting too flexible/complex)
- Will outperform lasso (generally) if all predictors have some relation to the response, in relatively equal proportion

A note on scaling

- While least squares models don't care about scale (coefficient estimates remain the same, but the intercept changes), some models are *heavily* impacted by the scale of the predictors
  - Intuition: Since the shrinkage penalty *literally depends on the size of the coefficients*, different scales can make a big impact
- Standard scaling is one option

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

- We can write a simple function to do this for us, or simply use the R function `scale` with provided defaults
- Fortunately, the `glmnet` function we'll use does scaling/standardization for us!

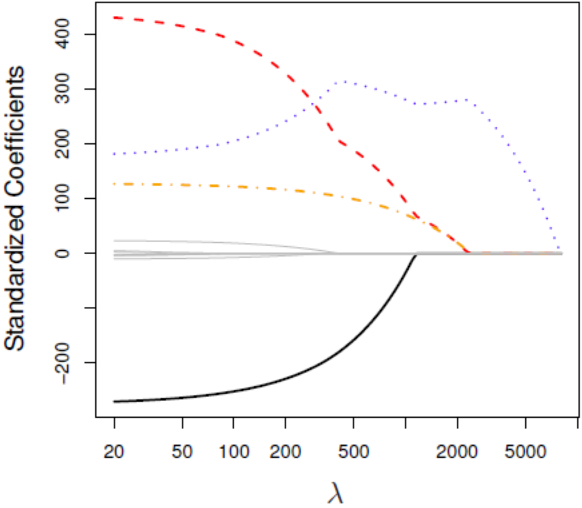
Lasso Regression

- Ridge regression keeps *all* predictors in the model, the *lasso* actually performs variable selection using a slightly different penalty

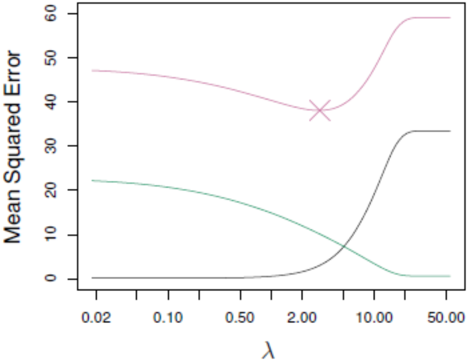
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} B_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

- This shrinks the estimates to zero as well, but actually forces some of those coeffients *to zero* when λ is high
- This has the desirable property of yielding *sparser* more interpretable models
- Again, choose *lambda* with cross-validation

Lasso Regression using Credit data



Lasso Regression using Credit data

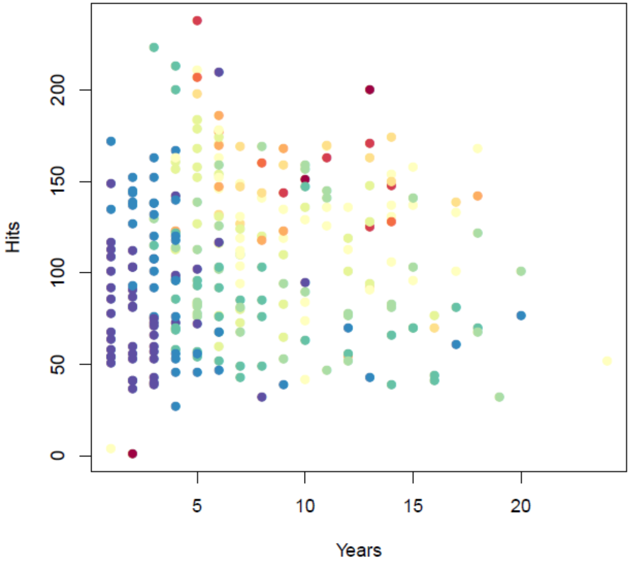


- Will outperform Ridge (generally) if only a *subset* of the predictors have a substantial impact on the response
- First-stage lasso can be valid approach for identifying a subset of relevant predictors out of a broader selection, especially in the face of hundreds of them

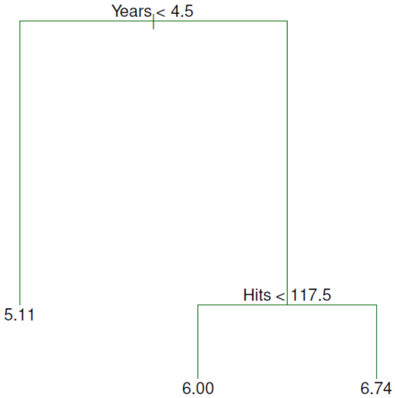
Decision Trees

- *Tree-based* methods involve segmenting the predictor space into subregions, and forming predictions based on how the space is split
  - The splitting rules resemble the branches of trees, hence decision trees
  - Not very powerful from a predictive standpoint
  - Simple and useful for interpretation, or understanding how a model might make decisions with your data
  - Mimics human thinking
- Tree analogy
  - Trees grow upside down (leaves on the bottom)
  - The leaves, or regions we divide into, are *terminal nodes*
  - Internal splits (the branches) are *internal nodes*

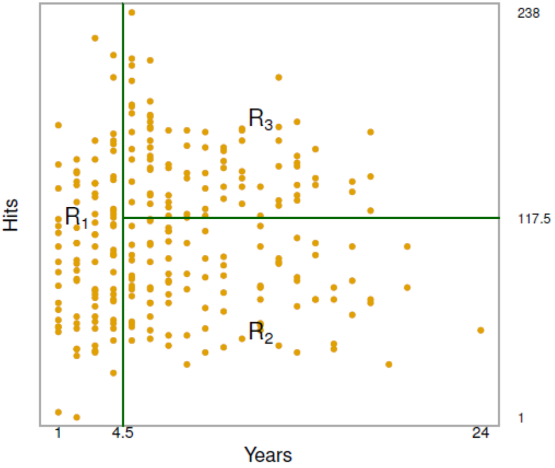
Baseball Salaries data example



Baseball Salaries data example



Baseball Salaries data example



And how!

Goal is to divide up the data into rectangular regions (boxes) that minimize RSS

- Uses a *top-down* (begins at the top of the tree and splits from there), *greedy* (makes the best choice at the current level, rather than entertaining subsequent possibilities) approach
- Continues until there are a minimum number of observations in each final region
- Predict the mean of the observations in a given region for observations that fall in that region
- Since this approach has a tendency to *overfit* the data, we apply a method called *cost complexity pruning* to essentially balance the tradeoff between having a very complex tree and how well the tree fits the training data
  - Since our goal is to do well on *test* data, we strike a balance using a tuning parameter α, kind of like λ in the lasso
  - We pick the best α with, you guessed it, cross-validation

Tree splitting algorithm

1. Use recursive binary splitting to grow a large tree until each terminal node contains some specified minimum number of observations
2. Use cost complexity pruning to obtain a sequence of better, smaller trees as a function of tuning parameter α
3. Use K-fold cross-validation to choose the optimal value of α
4. Return the subtree that corresponds to this optimal value of α

Classification trees

- *Very* similar to regression trees, except predicts each observation belongs to the *majority class* of the region to which it is assigned (as opposed to the average)
- Instead of using RSS as the metric by which we split our trees, other metrics are available, typically either *Gini index* or *cross-entropy*
- Gini index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- In simple terms, this is a measure of *node purity* - a small value here indicates that the node is "pure" and contains observations predominantly from a single class
- Other metrics are available - we won't discuss them here

Trees: Pros and Cons

- Super easy to explain and intuitive - likely even easier than linear regression
- More closely mimics human decision making than even OLS
- Small trees, in particular, can be displayed graphically and make sense to non-experts
- Handles categorical variables really well
- *Bad* predictive accuracy when applied out-of-sample (trees are highly susceptible to overfitting the specific data they are given)
    - Though, if your problem can't be modeled with a linear model, may do better

Bagging

- While individual trees are very limited, we can use *many* trees to improve performance, using a technique called *boostrap aggregation* or *bagging*
  - *Bootstrapping* involves generating repeated samples, selected with replacement, from a single training data set
  - Averaging a set of observations reduces variance
- Generate B bootstrapped samples:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- And simply average the predictions!
- High level: Build a bunch (hundreds or thousands) of trees on bootstrapped data, and either average the predictions (regression) or the majority vote (classification)

RandomForest

- Tweaks bagging to *decorrelate* trees and thereby reduce variance
- Just as before, build lots of decision trees on bootstrapped samples
- BUT each time we split, only a *random subset m* of the p predictors is considered
  - At each split in the tree, the algorithm *can't even consider a majority of the available*
  - A really strong predictor will dominate in bagging, but if it isn't even up for consideration in some of the random forest trees, it gives others a chance
  - This *decorrelates* the trees, and leads to further reduction in variance
- m here is a tuning parameter much like λ - we can choose an optimal m with cross-validation
- State-of-the-art algorithm for predictive modeling, but results are very difficult to interpret
  - We no longer have any concept of a standard error
  - But, randomForest can offer "variable importance" metrics

Deliverable

Using one of four cleaned (for the most part) data sets, build and evaluate a predictive model!

- Independent variables (targets) are specified; up to you what to include as predictors
- Make some justification for why you picked the predictors you did, and why you chose the model you did (i.e. logistic regression for classification tasks)
- Evaluate the model! We know that metrics from fitting the training data don't map well to fit out of sample
- Briefly explain your process and your findings
- OPTIONAL: Use some of the other models we discussed to further motivate your analysis

Send me an email with your R file, along with your explanation either in commented lines in the file, or in a separate text/word document file

References

An Introduction to Statistical Learning (James, Witten, Hastie, and Tibshirani) serves as the basis for the material herein http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf
http://faculty.marshall.usc.edu/gareth-james/ISL/